A bridge between logic and neural

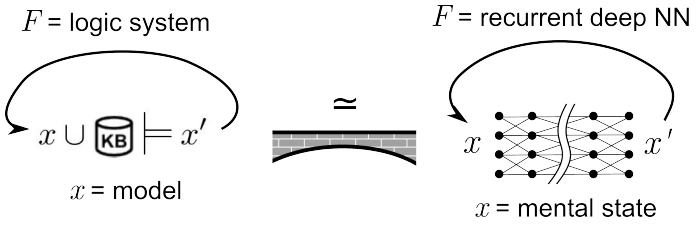
甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

Abstract. Logic-based inductive learning is based on slow combinatorial search — the bottleneck of classical AI. Neural network learning is based on gradient descent, which is much faster. We examine the abstract structure of logic (the consequence operator $\vdash$) and try to transfer this structure to the neural-network setting.

Keywords: neural-symbolic integration, logic, algebraic logic

This problem took a long time to solve because the structure of logic is not easily expressible as a mathematical structure:



(1)

1 The structure of neural networks

A **neural network** is basically:

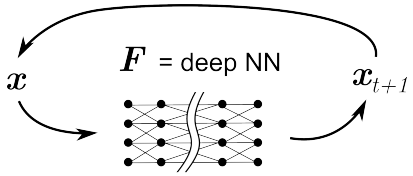
weight matrix for each layer total # of layers

$$F(\boldsymbol{x}) = \sigma(W_1\sigma(W_2...\sigma(W_L\,\boldsymbol{x}))) \tag{2}$$

where σ is the sigmoid function applied component-wise to each neuron (whose role is to provide **non-linearity**).

σ acts on each component of $\boldsymbol{x}$; Its action is **not invariant** under a change of coordinates. Therefore, σ is not a vector operation, and thus $\mathbb{X}$ is not a **vector space** structure. Common practice is to denote $\vec{x}$ as a vector, but this is misleading.

Consider, for example, to recognize *"white cat chases black cat"*. The *"cat"* object has to be recognied twice; Obviously we should not waste 2 neural networks to do this. If we connect a neural network **head to tail**, we get a minimalist cognitive architecture with a **recurrent** loop:



(3)

Its state equation is:

$$\boxed{\text{discrete time}} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t) \tag{4}$$

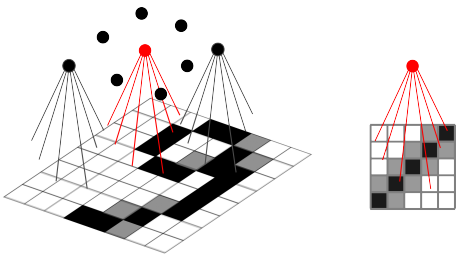
$$\boxed{\text{continuous time}} \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{5}$$

From this we can see that $\mathbb{X}$ is a **differentiable manifold** (cf. our 1st paper [13]).

1.1 What are "features"?

Now let's think about how a neural network performs pattern recognition, perhaps it would be illuminating....

Consider the simplest case, eg. a layer of neural network extracting visual features from the digit "9". This layer may have many neurons (left figure), each neuron locally covers a region of the input layer, the so-called "local receptive field" in the neuroscience of vision (right figure).



(6)

Suppose the red neuron is responsible for recognizing the "diagonal line" feature. Its equation is $\boldsymbol{y} = \mathcal{S}(W\boldsymbol{x})$. The matrix W's role is to affine "rotate" the feature space, so that the features we desire is pointing in a certain direction. Then we use $\mathcal{S}$ to "**squeeze**" the desired and undesired features. The output after $\mathcal{S}$ represents the **presence or absence** of a particular feature, ie $\{0, 1\}$. This is a form of information **compression**.

From the above reasoning, we may draw a general principle, which I call the **Neural Postulate**:

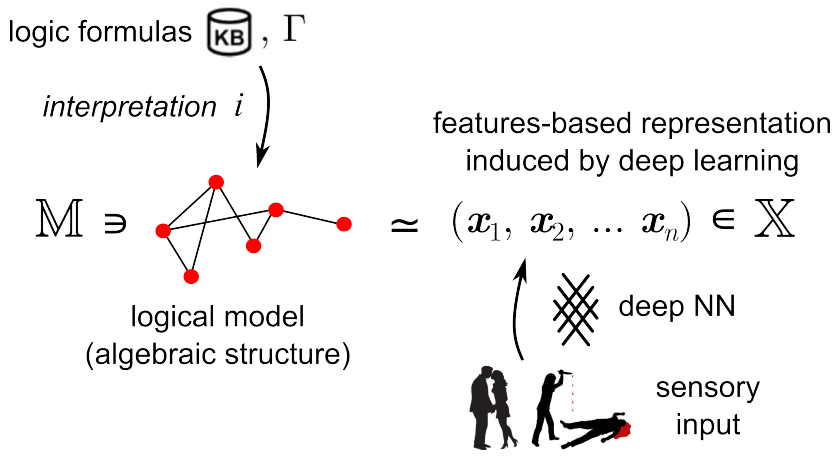
- Each neuron represents the presence or absence of a certain **feature**
- Higher-layer neurons represents **relations** between lower-layer features

(7)

Following this line of thinking, we may conjecture this correspondence:

$$\begin{array}{rcl} \mathcal{M} & \simeq & \mathcal{X} \\ \text{constant} \ \bullet & \Leftrightarrow & \text{neuron} \\ \text{relation} \ \bullet\!-\!\bullet & \Leftrightarrow & \text{relation between higher and lower neurons} \end{array} \tag{8}$$

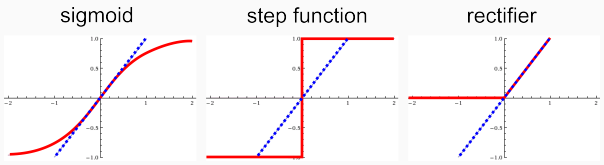
The following cartoon illustrates this correspondence from the perspective of **model theory**:



(9)

A little digression on chaos theory:

The role of $\int^{-1}$ is to "stretch", dragging points that are close neighbors to more distant positions. Looking at the common **threshold functions**, we see they are all non-linear **deformations** away from the identity $y = x$:



(10)

This is very similar to the "horseshoe" proposed by Steven Smale [9], a recipe for creating chaos. A variant of the Smale horseshoe is the "baker map", analogous to kneading dough in bakery. "Stretching" and then putting back to the original space, and repeating this process, creates chaos [2] [10].

2 Structure of logic

A **logic system** can be defined as:

- a set of symbols for **constants**, **predicates**, and **functions**
- **propositions** built from the above atoms
- **connectives** between simple propositions, eg: $\neg, \wedge, \vee$
- The **logical consequence** relation: $\Gamma \vdash \Delta$

The learning algorithm for logic-based AI is studied under the topic of ILP (inductive logic programming), which is a well-established field (cf. my tutorial [11] or de Raedt's textbook [7]). ILP performs **combinatorial search** in the symbolic space of logic formulas, and is very slow. The gradient descent of neural networks is much faster. In the following we examine the prospects of combining these 2 approaches.

Our idea is to transfer the structure of logic to neural networks. The logic structure can be broken down into:

- propositional-level structure
  (the mental state $\boldsymbol{x}$ is represented as a **set** of propositions $p_i \in \boldsymbol{x}$)
- sub-propositional structure

2.1 Propositional-level structure

2.1.1 Mental state = { propositions }

The mental state $\boldsymbol{x}$ may be broken down into a conjunction of propositions:

$$\boldsymbol{x}_1 = \text{ I'm attending a seminar } \wedge \text{ I feel hungry } \wedge .... \tag{11}$$

There could be another mental state:

$$\boldsymbol{x}_2 = \text{ I'm riding the subway } \wedge \text{ I feel hungry } \wedge .... \tag{12}$$

It would be very inefficient if x_1 and x_2 are considered completely different states (without decomposition).

2.1.2 Boolean algebra

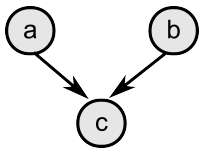
A **Boolean algebra** is a structure with:

$$\underbrace{\mathcal{B}}_{\text{underlying set}} = (\, A, \overbrace{\wedge, \vee}^{\text{binary ops}}, \underbrace{\neg}_{\text{unary op}} \,) \tag{13}$$

The most essential part of the Boolean structure is this **commutativity** relation:

$$\forall a, b \in \mathcal{B}. \qquad a \wedge b = b \wedge a \tag{14}$$

We may ignore the distinction between $a \wedge b$ and $a \vee b$ because they degenerate into the following Bayesian network if we use fuzzy-probabilistic truth values [12]:

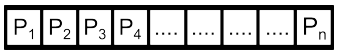


(15)

2.1.3 "Carousel" as a set of propositions

Our goal: embed a set of propositions (*order is unimportant*) as a vector. Suppose we know how to encode individual propositions as vectors, then the problem is how to encode a *variable* number of propositions as a longer vector.

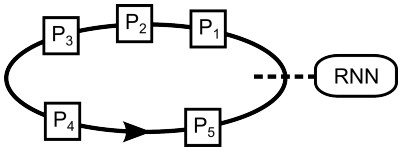
My first attempt is to fix the number of propositions to n, and impose a symmetry (commutativity) on the long vector, ie, that it be invariant under permutation of the "sub-vectors" (by the symmetric group S_n):



(16)

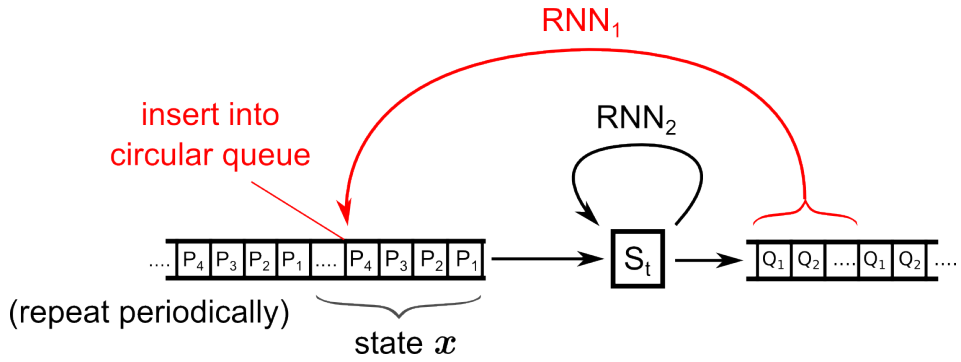
The vector space $\mathbb{X}$ then becomes a quotient $\mathbb{X}/S_n$ by the symmetry. Then the neural network can operate on this much-smaller quotient space. At first sight, this seems nice, but it is then realized that usually the neural network needs to detect the *presence* of a particular proposition among the P_i's, and that means it needs to detect the proposition on *each* position i. That off-sets the advantage gained by the reduction in size of the quotient space.

An alternative way to represent a set (that is easy for neural networks to processs) is to put its elements into a **carousel**:



(17)

For example, if we want to recognize the conjunction $a \wedge b \wedge c$, we use an RNN to *collect* these elements as they carousel in a circle. Note that the RNN for this carousel is not the same as the RNN in the "outer" architecture. Thus we get an **RNN-within-RNN** architecture:

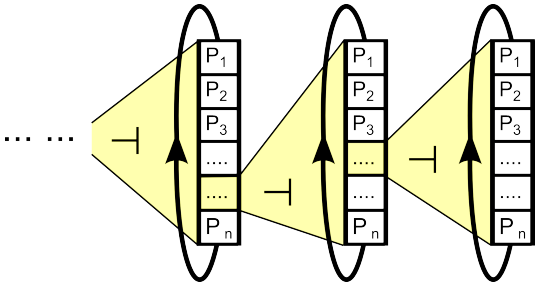


(18)

RNN_1 is the "outer" loop for processing the state-transition $\boldsymbol{x} \mapsto \boldsymbol{x}'$. RNN_2 is the "inner" loop for processing individual propositions within the state $\boldsymbol{x}$.

2.1.4 The structure of $\vdash$

Besides commutativity, another important aspect of $\vdash$ is that it usually does not change $\boldsymbol{x} \mapsto \boldsymbol{x}'$ completely: the premise may be some propositions in $\boldsymbol{x}$ but not all of $\boldsymbol{x}$, and the conclusion is usually just 1 or a few propositions that are added to $\boldsymbol{x}$:



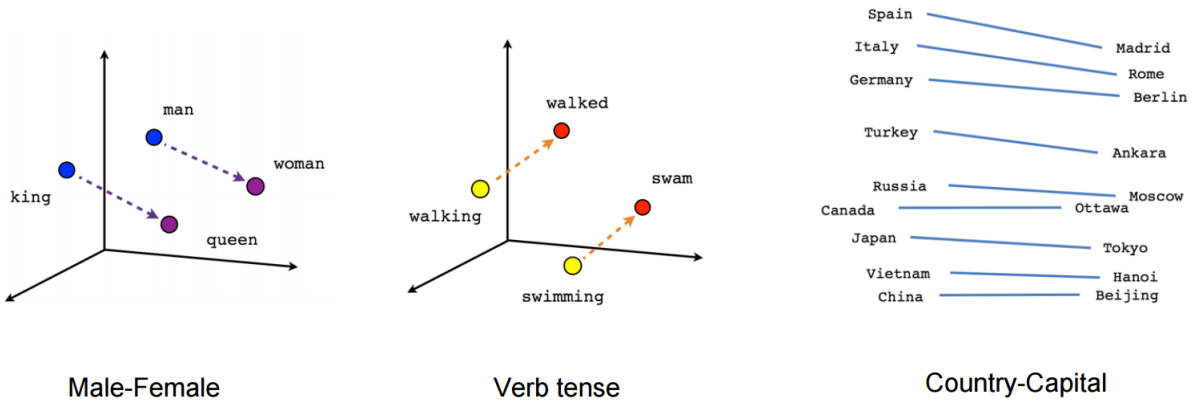
(19)

2.2 Sub-propositional structure

Traditionally the structure of predicate logic is rather esoteric from the point of view of mathematics, this has hindered the progress of logic and logic-based AI.

2.2.1 Semantic distance and failure of compositionality

Word2Vec[6] embeds words into vector space with "semantic distance":



(20)

After Word2Vec, a natural next step is to represent **sentences** with semantic distance. One popular approach (eg [1]) is via **tensor products**. For example:

$$\textit{"I love you"} \quad \Rightarrow \quad \mathrm{i} \otimes \mathrm{love} \otimes \mathrm{you} \tag{21}$$

but this approach has serious problems: For example, pronouns such as "you" refer to various entities and must be resolved using rather complicated procedures, known as "abductive interpretation" in logic-based AI. Without such interpretations the word-entity correspondence becomes very inaccurate.

2.2.2 A logic based on "features"

Assuming that the compositionality problem can be bypassed, perhaps with logical atoms referring directly to entities (no pronouns, metaphors, etc), we may represent sentences by a "conjunction" of features (different from the $\wedge$ for propositions), eg:

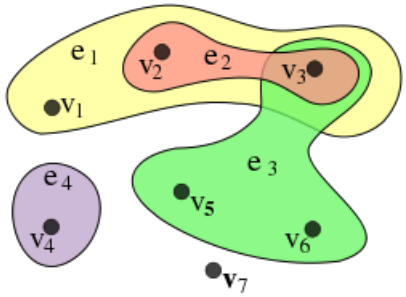
$$\begin{aligned} \boldsymbol{x} &= \textit{I feel hungry} \\ &= \textit{me} \cap \textit{physiology} \cap \textit{food} \cap \textit{negative-emotion} \cap \ldots. \end{aligned} \tag{22}$$

This is like using a series of binary choices (= features) to determine an idea, like the game "20 Questions".

2.2.3 OpenCog's logic based on hypergraphs

To represent a complex scenario with many inter-relations, without using pronouns, perhaps **hypergraph** is a good choice. This is the idea behind OpenCog's representation.

A hypergraph is also called a "set system". Any hypergraph can be represented as a set of edges, each edge being an element of the power set of the nodes. Eg:



(23)

This representation is convenient for our purpose: Each edge can be represented as 1 proposition. The mental state $\boldsymbol{x}$ = a hypergraph = a set of propositions. We can use the symmetric space proposed above to represent the mental state.

Each proposition (edge) is a product of atomic concepts (nodes), we may further fix #(nodes) per edge to, say, 3:

$$\boxed{\text{proposition}} \quad \boldsymbol{p} = c_1\ c_2\ c_3 \tag{24}$$

For example $c_1\ c_2\ c_3$ could be like subject-verb-object in natural language, where c_1 and c_3 are from the set of nouns / entities; c_2 from the set of verbs / relations. We don't need to be very precise about this, as the actual representation can be figured out via machine learning — in the **post-structuralist** view of AI, as opposed to the structuralist view, where structural elements are designated with labels such as "NP (noun phrase)" in phrase-structure grammar or the "is-a" link in old-fashioned AI.

2.2.4 "Linkage" phenomena in predicate logic

Next we consider **variable substitution**, a key feature of predicate logics. For example:

$$\forall X\ \forall Y\ \forall Z.\quad \text{grandfather}(X, Z) \leftarrow \text{father}(X, Y) \wedge \text{father}(Y, Z) \tag{25}$$

The links symbolize that the same variables must be substituted by the same objects, which is really the *essence* of **substitution**.

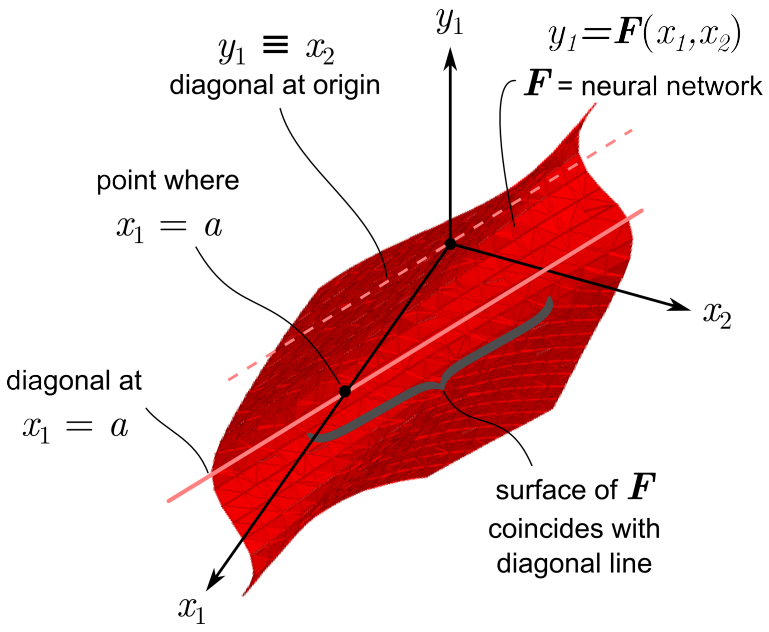
To duplicate this effect in neural networks, we can force a coordinate of the input space to be identified with another coordinate of the output space:

$$\boldsymbol{F}:\ (x_1, ..., x_i, ..., x_n) \overset{id}{\mapsto} (y_1, ..., y_j, ..., y_n) \tag{26}$$

But such identities do not always hold. They only exist at specific values of $\boldsymbol{x}$:

$$\boldsymbol{F}: \begin{cases} y_j \equiv x_i, & \text{if } \boldsymbol{x} = \hat{\boldsymbol{a}} \text{ for some coordinates except } x_i \\ y_h \equiv x_k, & \text{if } \boldsymbol{x} = \hat{\boldsymbol{b}} \text{ for some coordinates except } x_k \\ \text{... etc ...} & \\ \text{is free otherwise} & \end{cases} \tag{27}$$

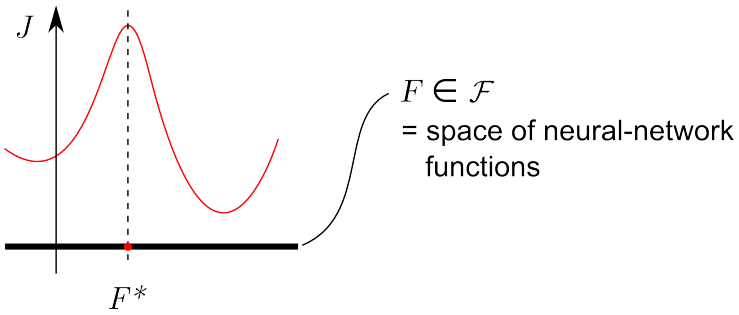
Geometrically, a linkage is a diagonal line that *restricts* the "graph" of the neural network function:



(28)

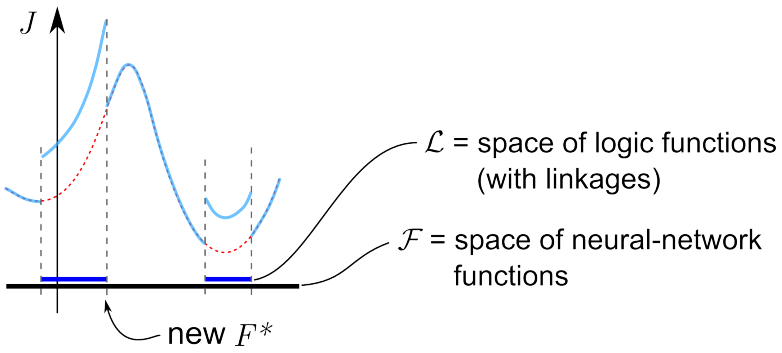
2.2.5 Learning algorithm with linkages

As an optimization problem, the goal of learning is to find the optimal neural network F^* in a function space (figuratively):



(29)

We need a new algorithm that gives priority (higher scores) to linkages:



(30)

As for learning algorithm, one idea is to *force* the learning of diagonals, whenever a potential identity function is detected. This creates a "propensity" for generalization. After a diagonal is learned, it may be destroyed by subsequent learning; which is fine. Also, the forcing of a diagonal may accidentally

destroy previously learned knowledge. This may be balanced by **dual-lobe** re-normalization.

```
for each data item do
    detect if a potential identity may exist;
    if identity seems to exist then
        force-learn the diagonal;
    else
        learn as usual;
    end
end
dual-lobe re-normalization;
```
(31)

2.3 Algebraic logic

Traditionally, there are 2 ways to express the **sub-propositional** structure of logic: via **predicate logic** and **relation algebra**. No matter which way, their essence is **variable substitution**. Interestingly, substitution is also the essence of "algebra", but here substitutions occur **implicitly**, so it is actually difficult to use algebra to express them explicitly.

As for **λ-calculus**, it is essentially also a scheme for managing substitutions. Whereas **combinatory logic**, equivalent to λ-calculus, eliminates the use of "variables". The price to pay is an increase in length of the formulas (more complex than relation algebra).

The algebraization of (first-order) predicate logic is given by Alfred Tarski's **cylindric algebra**:

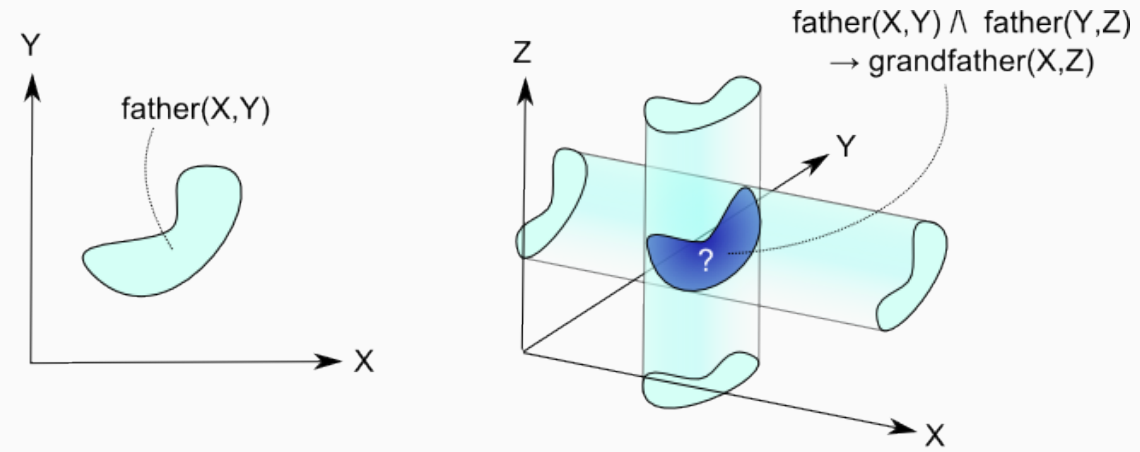
$$\frac{\text{propositional calculus}}{\text{Boolean algebra}} = \frac{\text{predicate calculus}}{\text{cylindric algebra}} \tag{32}$$

The algebraization of **higher-order logic** (HOL) is provided by **topoi theory** [3] [4]:

$$\frac{\text{intuitionistic propositional calculus}}{\text{Heyting algebra}} = \frac{\text{higher-order logic}}{\text{elementary topos}} \tag{33}$$

HOL is equivalent to **untyped** λ-calculus, whereas the **typed** λ-calculus (= type theory) is somewhat weaker. The characteristic of the entire HOL family is "substitution of equal by equals", which causes the length of formulas to increase. But predicate logic performs substitution with **objects**, so it does not increase the length of formulas; However, its algebraization introduces notions such as cylindrification and the diagonal set. For example, in fig. (28) there is the diagonal $y_1 \equiv x_2$.

This is an illustration of **cylindrification**:



(34)

The intersection of 2 cylinders can be very irregular. For two L-shaped cylinders, the intersection is like this:

2.4 Relation algebra

I used to think **relation algebra** (RA) [8] [5] is a promising knowledge representation language, for it seems close to natural language in some aspects, when in fact RA and first-order logic (FOL) are basically equivalent. In FOL there is the complexity of linkages, whereas in RA this complexity has not disappeared; We could say "complexity is conserved" [a].

For example, "*father's father is grand-father*" in (25) can be expressed easily in RA:

$$\text{father} \circ \text{father} = \text{grandfather} \tag{35}$$

The actual inference steps are:

$$\begin{aligned} \text{John father Pete} \\ \text{Pete father Paul} \\ \text{John father} \circ \text{father Paul} \end{aligned} \tag{36}$$

Then we need to perform a **substitution** to get the conclusion:

$$\text{John grandfather Paul} \tag{37}$$

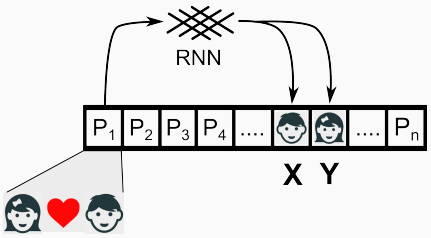
So the complexity of linkages becomes the complexity in *formula length*.

2.5 Fractal structure

While "size" is not an issue of itself, if our goal is to embed logic formulas into the state $\boldsymbol{x}$, then a *variable* number of formulas does pose a problem. If we must put a variable number of formulas into a (finite-dimensional) vector space, it seems that some sort of self-similar or **fractal structure** is required, and we also need a new kind of neural network that *a priori*ly has fractal structure; but this seems rather complicated and we have not explored further.

2.6 Concrete substitution

We described the geometric structure of linkages in substitutions, but we can also view substitution as a **multi-step** process: by storing some information into memory "boxes", such boxes serve as **variables**, concretely carrying out the substitution. In other words, converting **space complexity** into **time complexity**[b]; below is a conceptual diagram:



(38)

X and Y are "boxes" (= variables) in the state $\boldsymbol{x} \in \mathbb{X}$.

The linkage structure mentioned earlier is easy to use in finite-dimensional vector spaces, but it is first-order logic, which has limitations when dealing with **higher-order** relations. For the latter, the multi-step method may be a good solution.

[a] This is a quote from the category theorist Eugenia Cheng.
[b] Human brain waves occur in various frequencies from 0.5 Hz to 40 Hz. What we percieve as instantaneous in our mind may be the result of many iterations of basic steps.

2.7 Cartesian-closedness

We still lack a crucial property: **Cartesian-closedness**. This means that in a certain category, for 2 arbitrary objects A, B, we could always find their:

$$\boxed{\text{product}}\ A \times B \quad \text{and} \quad B^A\ \boxed{\text{exponentiation}} \tag{39}$$

In logic this refers to:

$$A \wedge B \quad \text{and} \quad A \to B \tag{40}$$

Why do we need Cartesian-closedness? Notice that in our minimal architecture we only have 2 types of memory: the **instantaneous** $\boldsymbol{x}$ and the **permanent** $\boldsymbol{F} = \text{KB} \vdash$. From the logic-based AI point of view, KB stores **logic rules**, whereas $\boldsymbol{x}$ contains only **ground facts**. **Ground sentences** are formulas **without variables**, for example:

$$\boldsymbol{x} = \textit{seeing the floor has blood stains} \tag{41}$$

In contrast, **logic rules** are *conditional statements* that contain variables, eg:

$$\forall Z. \quad Z \textit{ has blood stain} \to Z \textit{ may be a crime scene} \tag{42}$$

If we want to store a rule inside $\boldsymbol{x}$, that means $\boldsymbol{x} \ni \mathbb{X}$ is a Cartesian-closed category. Such an $\mathbb{X}$ is a more powerful structure, that allows the system to think more complex thoughts. More importantly, $\boldsymbol{x} \ni \mathbb{X}$ and $F = \text{KB} \vdash$ are now on **equal footing** because they are both $\mathbb{X} \to \mathbb{X}$ **functions**, since Cartesian-closed implies $\mathbb{X} \simeq \mathbb{X}^{\mathbb{X}}$. This property is very useful in **belief revision** (see below).

How can a neural network be Cartesian-closed? Recall that in a neural network $\boldsymbol{x}$ is a set of **features** $= (x_1, x_2, ..., x_n)$. One method is to change all $\boldsymbol{x}$'s into functions $\mathbb{X} \to \mathbb{X}$. The logical **implication** $A \to B$ is obviously a function, but a single ground sentence A can also be turned into a function, as $\top \to A$, where $\top$ = logical "true". Note: These functions may contain *linkages*, ie, the ability to deal with variables.

As neural networks, $A \to B$ is a neural network $\mathbb{X} \to \mathbb{X}$. $\top \to A$ is also a neural network $\mathbb{X} \to \mathbb{X}$, but it maps $\mathbf{1} \mapsto \boldsymbol{A}$.

So what kind of space is $\mathbb{X}$? It could be the **weights** of a deep-learning network, but this NN's input/output layer must have the **width** to process **itself** ! This seems impossible. At first I tried to *restrict* $\boldsymbol{F}(\boldsymbol{x})$ to the mapping $a \wedge b \wedge c \wedge .... \to d$, or in other words, $\boldsymbol{F}|_{\boldsymbol{x}=a\wedge b\wedge c...}$. Such "**micro-functions**" may be smaller in size than the whole $\boldsymbol{F}$. But a **multi-layer** neural network is a very peculiar machine in the sense that its classification of the source domain is dependent on *every* layer of weights, in other words, its components are intertwined and the mapping as a whole *cannot be decomposed* easily.

At this time the only solution I can think of is to "clamp" the 2 sides of $a \wedge b \wedge c \wedge .... \to d$ to $\boldsymbol{F}(\boldsymbol{x})$ and force it to reproduce this input-output pair. This has the effect that the content of $\boldsymbol{x}$ gets *transferred* into $\boldsymbol{F}$.

2.8 Application to belief revision

Belief revision (also known as "truth maintenance") is a high-water mark of classical logic-based AI. If our new architecture can perform belief revision, we can be fairly assured that it has general intelligence.

Normally, a logical operation is the KB acting on $\boldsymbol{x}$ to yield a new $\boldsymbol{x}$:

$$\text{KB}(\boldsymbol{x}) = \boldsymbol{x}' \tag{43}$$

Whereas belief revision can perhaps be viewed as $\boldsymbol{x}$ acting on KB:

$$\boldsymbol{x}(\text{KB}) = \text{KB}' \tag{44}$$

Because of Cartesian-closedness, $\boldsymbol{x}$ and KB have equal status, that makes the above operation possible.

This is just a vague idea, I shall return to fill in this blank....

Acknowledgement

Thanks to Ben Goertzel for discussions on the AGI mailing list. Ben first pointed out the advantage of neural network learning over inductive logic learning, which prompted me to research their relationships. Thanks also to Juan Carlos Pinto for personal discussions on the structure of neural networks.

References

1. Coecke, Sadrzadeh, and Clark. Mathematical foundations for distributed compostional model of meaning. *Linguistic Analysis 36, 345-384*, 2010.
2. Robert Gilmore and Marc Lefranc. *The topology of chaos: Alice in stretch and squeezeland.* Wiley-VCH, 2011.
3. Joachim Lambek. Categorical versus algebraic logic. In Andréka, Monk, and Németi, editors, *Algebraic logic*, pages 351–360. North-Holland, 1988.
4. Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic – a first introduction to topos theory.* Springer, 1992.
5. Roger Maddux. *Relation algebras.* Elsevier, 2006.
6. Mikolov, Sutskever, Chen, Corrado, and Dean. Efficient estimation of word representations in vector space. *Proceedings of workshop at ICLR*, 2013.
7. Luc De Raedt. *Logical and relational learning.* Springer, 2008.
8. Gunther Schmidt. *Relational mathematics.* Cambridge, 2010.
9. Stephen Smale. Differentiable dynamical systems. *Bulletin of the American Mathematical Society*, 1967.
10. Tamás Tél and Márton Gruiz. *Chaotic dynamics: an Introduction based on classical mechanics.* Cambridge, 2006.
11. King Yin Yan. ILP tutorial `https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/genifer/Genifer-induction(30July2012).pdf`.
12. King-Yin Yan. Fuzzy-probabilistic logic for common sense reasoning. *Artificial general intelligence 5th international conference, LNCS 7716*, 2012.
13. King Yin Yan, Juan Carlos Kuri Pinto, and Ben Goertzel. Wandering in the labyrinth of thinking – a cognitive architecture combining reinforcement learning and deep learning. (to be submitted AGI-2017).